A SIMPLE UNIVERSAL PATTERN-MATCHING AUTOMATON

DANIEL J. BERNSTEIN

Abstract. Consider an infinite non-deterministic automaton with one state \( p \) for each regular expression \( p \); transitions \( q \xrightarrow{c} qS \) whenever \( S \) is a character set containing \( c \); and null transitions \( q \xrightarrow{} q \), \( qr \xrightarrow{} qr \), \( qr \xrightarrow{r} q(r' \cup r) \), and \( qr' \xrightarrow{} q(r' \cup r) \). If this automaton starts at the empty regular expression, then \( p \) recognizes exactly the language defined by \( p \), for every \( p \). The subautomaton affecting \( p \) has at most \( 1 + \text{len} \ p \) states.

1. Introduction

This paper presents a nondeterministic infinite automaton that recognizes all regular expressions simultaneously. A portion of the automaton is shown in Figure 1 below.

The automaton has one state \( p \) for each regular expression \( p \), and no other states. The language recognized by \( p \) is exactly the language defined by \( p \). The subautomaton affecting \( p \) has at most \( 1 + \text{len} \ p \) states. Here \( \text{len} \ p \) is the number of non-parenthesis symbols in \( p \); for example, the length of \( ?xyxz \) is 7, and the length of \( ((xy \cup z)z) \cup yyy \) is 9.

Is it surprising that such an automaton exists? Of course not. It is well known that, for each \( p \), there is a nondeterministic automaton recognizing \( p \) with at most \( 1 + \text{len} \ p \) states. One can mechanically assign to each state a corresponding regular expression, and finally merge the automata for all \( p \) into a single infinite automaton that behaves as described above.

What is surprising about the automaton in this paper is that its definition is extremely short. There is one transition \( q \xrightarrow{} qC \) for each regular expression \( q \), character \( c \), and character set \( C \) containing \( c \). There are null transitions \( q \xrightarrow{} q \), \( qr \xrightarrow{} q(r' \cup r) \), and \( qr' \xrightarrow{} q(r' \cup r) \), for all regular expressions \( q, r, r' \). The automaton begins at \( () \) where \( () \) is the empty pattern. That’s it.

These transitions are visibly correct, in the sense that any string recognized by \( p \) is in the language defined by \( p \). It is not as easy to prove that these transitions are adequate, in the sense that any string in the language defined by \( p \) is recognized by \( p \). See Theorem 4.1 and Theorem 4.2. It also takes some work to prove that the subautomaton affecting \( p \) has at most \( 1 + \text{len} \ p \) states. See Theorem 7.2. These proofs occupy the remaining sections of this paper.

Date: 20000806.

2000 Mathematics Subject Classification. Primary 68Q45.

Key words and phrases. Regular expressions, nondeterministic infinite automata.

The author was supported by the National Science Foundation under grant DMS-9600083.
Notation and terminology. The string \((c_1, \ldots, c_n)\) is abbreviated as “\(c_1 \ldots c_n\)”. For example, “\(x\)”, “\(xyz\)”, and “\(xyzzy\)” are strings over the alphabet \(\{x, y, z\}\).

A **constant** is a set of single-character strings. The set of all single-character strings is denoted \(?\). At the risk of confusion, I abbreviate the constant \(\{c\}\) as \(c\) for each character \(c\).

A **pattern algebra** is a set with an associative binary operation “composition” written \(p, q \mapsto pq\), a neutral element for composition written \(()\), a unary operation “closure” written \(p \mapsto p\), and a binary operation “union” written \(p, q \mapsto p \cup q\). For example, the set of regular languages is a pattern algebra with composition \(L, M \mapsto LM = \{st : s \in L, t \in M\}\); neutral element \((() = \{\:\})\); union equalling the usual set union; and closure \(L \mapsto \overline{L} = L^0 \cup L^1 \cup \cdots\).

A **pattern** is an element of the free pattern algebra on the set of constants. In other words, a pattern is a formula built up from constants via union, closure, and composition, modulo redundant parentheses and the associativity of composition. Every pattern falls into one of the four forms \((\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\\)\).

I write \(s \in p\), and say \(p\) **matches** \(s\), to mean that \(s\) is contained in the language defined by \(p\). Here \(p\) is a pattern and \(s\) is a string.

Historical notes. Thompson in [1] constructed an automaton recognizing \(p\) with \(O(\text{len } p)\) states. See [2] for a coherent survey of subsequent constructions. My construction is simpler than any of the constructions in [2].

I wrote down my automaton in June 1991, and distributed an implementation in a posting to alt.sources in January 1992. I was not familiar with the literature; the construction seemed so obvious that I assumed it was what everyone had always done. In April 1994, after reading a preliminary version of the taxonomy in [2], I announced my automaton in a posting to comp.theory.

2. Pattern implication

The relation “\(p' \Rightarrow p\)” is the transitive closure of the **basic implications** “\(q \Rightarrow q\overline{r}\)”, “\(q\overline{r} \Rightarrow q\overline{r}\)”, “\(qr \Rightarrow q(r' \cup r)\)”, and “\(qr' \Rightarrow q(r' \cup r)\)”. In other words, \(p' \Rightarrow p\) if and only if there is a chain \(p' = p_0 \Rightarrow p_1 \Rightarrow \cdots \Rightarrow p_n = p\) of basic implications. In particular \(p \Rightarrow p\).

For example, \((\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\\)\). and \(\overline{?} \Rightarrow \overline{?xy}\), so \((\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\\)\). and \(\overline{?} \Rightarrow \overline{?xy}\).  

**Theorem 2.1.** If \(p' \Rightarrow p\) and \(s \in p'\) then \(s \in p\).
Proof. It suffices to check the basic implications: if \( s \in q \) or \( s \in q\overline{r} \) then \( s \in q\overline{r} \); and if \( s \in qr \) or \( s \in qr' \) then \( s \in q(r' \cup r) \).

\( \square \)

**Theorem 2.2.** If \( p' \Rightarrow p \) then \( p''p' \Rightarrow p''p \).

Proof. This is a purely formal consequence of the definition: \( p''q\overline{r} \Rightarrow p''q, p''q \Rightarrow p''q\overline{r}, p''qr \Rightarrow p''q(r' \cup r) \), and \( p''q\overline{r} \Rightarrow p''q(r' \cup r) \), so given a chain of basic implications we may prepend \( p'' \) to each term.

\( \square \)

### 3. The \( \text{impl} \) function

Define recursively

\[
\text{impl } p = \begin{cases} 
\{p\} & \text{if } p = () \text{ or } p = qC, \\
\text{impl } qr' \cup \text{impl } qr & \text{if } p = q(r' \cup r), \\
\text{impl } q \cup \{q\overline{r}y : y \in \text{impl } r, y \neq ()\} & \text{if } p = q\overline{r}.
\end{cases}
\]

For example, \( \text{impl } ?xxy = \{((), ??, \overline{x}xxy)\} \).

**Theorem 3.1.** If \( x \in \text{impl } p \) then \( x \Rightarrow p \).

Proof. Induct on \( \text{len } p \). First, if \( p = () \) or \( p = qC \) then \( x = p \) so \( x \Rightarrow p \).

Second, say \( p = q(r' \cup r) \). If \( x \in \text{impl } qr' \) then by induction \( x \Rightarrow qr' \Rightarrow p \). If \( x \in \text{impl } qr \) then by induction \( x \Rightarrow qr \Rightarrow p \).

Third, say \( p = q\overline{r} \). If \( x \in \text{impl } q \) then by induction \( x \Rightarrow q \Rightarrow p \). If \( x = q\overline{r}y \) with \( y \in \text{impl } r \), then by induction \( y \Rightarrow r \), so \( x = q\overline{ry} \Rightarrow q\overline{r} \Rightarrow p \) by Theorem 2.2.

\( \square \)

**Theorem 3.2.** If \( x \in \text{impl } p \) then \( x \) is empty or ends with a constant.

Proof. Induct on \( \text{len } p \). First, if \( p = () \) or \( p = qC \) then \( x = p \).

Second, if \( p = q(r' \cup r) \) then \( x \in \text{impl } qr' \) or \( x \in \text{impl } qr \); by induction \( x \) is empty or ends with a constant.

Third, say \( p = q\overline{r} \). If \( x \in \text{impl } q \) then by induction \( x \) is empty or ends with a constant. If \( x = q\overline{ry} \), with \( () \neq y \in \text{impl } r \), then by induction \( y \) ends with a constant, so \( x \) does too.

\( \square \)

**Theorem 3.3.** If \( s \in p \) then \( s \in x \) for some \( x \in \text{impl } p \).

Proof. Induct on \( \text{len } p \). First, if \( p = () \) or \( p = qC \) then \( p \in \text{impl } p \) so there is nothing to prove.

Second, if \( p = q(r' \cup r) \) then \( s \in qr' \) or \( s \in qr \). So by induction \( s \in x \) where \( x \in \text{impl } qr' \) or \( x \in \text{impl } qr \). Either way \( x \in \text{impl } p \).

Third, if \( p = q\overline{r} \) then \( s = t't \) with \( t' \in q \), \( t \in \overline{r} \). If \( t = "" \) then \( s = t' \in q \) so by induction there is an \( x \in \text{impl } q \) with \( s \in x \); and \( x \in \text{impl } p \). If \( t \neq "" \) we may write \( t = uv \) where \( u \in \overline{r} \) and \( v \in r \), \( v \neq "" \). By induction there is a \( y \in \text{impl } r \) with \( v \in y \); since \( v \) is nonempty we cannot have \( y = () \). Thus \( q\overline{ry} \in \text{impl } p \), and \( s = t'uv \in q\overline{ry} \) as desired.

\( \square \)

### 4. Consequences of \( \text{impl} \)

Theorem 4.1 says that my automaton works for the empty string. Theorem 4.2 says that, if my automaton works for a string \( t \), then it also works for \( t \) plus any character.

**Theorem 4.1.** "" \in p if and only if () \Rightarrow p.""
Proof. Say \((\cdot) \Rightarrow p\). Certainly \("\cdot\) \in \cdot\); by Theorem 2.1, \("\cdot\) \in p\.

Conversely, say \("\cdot\) \in p\). By Theorem 3.3, \("\cdot\) \in x\) for some \(x \in \text{impl} p\). Since \("\cdot\) \in x\), \(x\) cannot end with a constant; thus, by Theorem 3.2, \(x = (\cdot)\). By Theorem 3.1, \((\cdot) = x \Rightarrow p\).

\[\text{Theorem 4.2. } t"c" \in p \text{ if and only if there exist } q \text{ and } C \text{ such that } t \in q, \"c\" \in C, \text{ and } qC \Rightarrow p.\]

\[\text{Proof. If } t \in q \text{ and } \"c\" \in C \text{ then } t"c" \in qC; \text{ if also } qC \Rightarrow p \text{ then } t"c" \in p \text{ by Theorem 2.1.} \]

Conversely, say \(t"c" \in p\). By Theorem 3.3, \(t"c" \in x\) for some \(x \in \text{impl} p\). Since \(t"c" \neq "\cdot\), \(x\) cannot be empty; thus, by Theorem 3.2, \(x\) ends with a constant. Write \(x = qC\). Then \(t \in q\) and \("c\) \in C\). Finally \(qC = x \Rightarrow p\) by Theorem 3.1.

\[5. \text{ Left subpatterns}\]

The relation \("p'\) is a left subpattern of \(p\)" is the transitive closure of the relations \("q\) is a left subpattern of \(qC\), \("q\r r\) is a left subpattern of \(q\r\), \("q\) is a left subpattern of \(q\r\)\), \("q\r\) is a left subpattern of \(q'(r' \cup r')\), and \("q\r\) is a left subpattern of \(q'(r' \cup r)\)\).

In other words, \(p'\) is a left subpattern of \(p\) if and only if \([p']\) affects \([p]\) in my automaton. Therefore, if \(s \in p\), then any prefix \(s'\) of \(s\) satisfies \(s' \in p'\) for some left subpattern \(p'\) of \(p\).

Example: \(?\) is a left subpattern of \(\overline{?}\), which is a left subpattern of \(\overline{?}\overline{xy}\), which is a left subpattern of \(\overline{?}\overline{xy}x\), which is a left subpattern of \(\overline{?}\overline{xy}xz\). So \(?\) is a left subpattern of \(\overline{?}\overline{xy}xz\).

\[6. \text{ The left function}\]

Define \(\text{left}_0 p = p\). Define \(\text{left}_{n+1} p\) for each \(n\) as follows:

\[\text{left}_{n+1} p = \begin{cases} \text{undefined} & \text{if } p = () \\ \text{left}_n q & \text{if } p = qC \\ \text{left}_n q\r r & \text{if } p = q\r, n < \text{len} r \\ \text{left}_{n-\text{len} r} q & \text{if } p = q\r, n \geq \text{len} r \\ \text{left}_n q r & \text{if } p = q(r' \cup r), n < \text{len} r \\ \text{left}_{n-\text{len} r} q r' & \text{if } p = q(r' \cup r), n \geq \text{len} r. \end{cases}\]

For example, the values of \(\text{left}_n \overline{?}\overline{xy}xz\), as \(n\) increases from 0 through 7, are \(\overline{?}\overline{xy}xz, \overline{?}\overline{xy}x, \overline{?}\overline{xy}, \overline{?}\overline{xy}xy, \overline{?}\overline{xy}x, \overline{?}, ??, \) and \(()\). For \(n > 7\), \(\text{left}_n \overline{?}\overline{xy}xz\) is undefined.

The reader may enjoy verifying certain facts not needed in this paper: \(\text{left}_{\text{len} p} p = ()\); if \(n < \text{len} p\) then \(\text{left}_n p\) is defined and nonempty; if \(n \leq \text{len} p\) then \(\text{left}_n p' p = p' \text{left}_n p\); if \(\text{left}_n p\) is defined then it is a left subpattern of \(p\).

\[\text{Theorem 6.1. } \text{If } n > \text{len} p \text{ then } \text{left}_n p \text{ is undefined.}\]

\[\text{Proof. Induct on } n. \text{ Note that } n > 0.\]

If \(p = ()\) then \(\text{left}_n p\) is undefined. If \(p = qC\) then \(n > n - 1 > \text{len} q\) so \(\text{left}_{n-1} q\) is undefined by induction; so \(\text{left}_n p = \text{left}_{n-1} q\) is undefined. If \(p = q\r\) then \(n > n - 1 - \text{len} r > \text{len} q\) so \(\text{left}_{n-1-\text{len} r} q\) is undefined by induction; so \(\text{left}_n p = \text{left}_{n-1-\text{len} r} q\) is undefined. If \(p = q(r' \cup r)\) then \(n > n - 1 - \text{len} r > \text{len} q r'\) so \(\text{left}_{n-1-\text{len} r} q r'\) is undefined by induction; so \(\text{left}_n p = \text{left}_{n-1-\text{len} r} q r'\) is undefined.
Theorem 6.2. left\(_n\) p'p = left\(_{n-\text{len}}\) p'p' if \(n \geq \text{len} p\).

Proof. Induct on \(n\).

If \(p = ()\) then \(\text{len} p = 0\) and \(p'p = p'\) as desired. If \(p = qC\) then \(n > n - 1 \geq \text{len} q\) so

\[
\text{left}_n p'p = \text{left}_{n-1} p'q = \text{left}_{n-1} \text{len}_q p'p = \text{left}_{n-1} \text{len}_p p'
\]

by induction. If \(p = q\bar{r}\) then \(n > n - 1 - \text{len} r \geq \text{len} q\) so

\[
\text{left}_n p'p = \text{left}_{n-\text{len} r} p'q = \text{left}_{n-\text{len} r-\text{len} q} p' = \text{left}_{n-\text{len} p} p'
\]

by induction. If \(p = q(r' \cup r)\) then \(n > n - 1 - \text{len} r \geq \text{len} q r'\) so

\[
\text{left}_n p'p = \text{left}_{n-\text{len} r} p'q r' = \text{left}_{n-\text{len} r-\text{len} q r'} p' = \text{left}_{n-\text{len} p} p'
\]

by induction. \(\square\)

7. The \(J\) Function

Theorem 7.2 states that all left subpatterns of \(p\) are values of \(\text{left}_n p\) for \(n \in \{0, 1, \ldots, \text{len} p\}\). Thus the number of states affecting \(\lceil p \rceil\) in my automaton is at most \(1 + \text{len} p\).

Define \(J(p, m, 0) = m\). For each \(n\) define \(J(p, m, n + 1) =\)

\[
\begin{cases}
\text{undefined} & \text{if } p = () \\
J(q, m, n) + 1 & \text{if } p = qC \\
J(q\bar{r}r, m, n) + 1 & \text{if } p = q\bar{r}, n < \text{len} r, J(q\bar{r}r, m, n) < \text{len} r \\
J(qr, m, n) - \text{len} r & \text{if } p = q\bar{r}, n < \text{len} r, J(q\bar{r}r, m, n) \geq \text{len} r \\
J(q, m, n - \text{len} r) + \text{len} r + 1 & \text{if } p = q\bar{r}, n \geq \text{len} r \\
J(qr, m, n) + 1 & \text{if } p = q(r' \cup r), n < \text{len} r, J(qr, m, n) < \text{len} r \\
J(qr, m, n) + \text{len} r' + 1 & \text{if } p = q(r' \cup r), n < \text{len} r, J(qr, m, n) \geq \text{len} r \\
J(qr', m, n - \text{len} r) + \text{len} r + 1 & \text{if } p = q(r' \cup r), n \geq \text{len} r.
\end{cases}
\]

Theorem 7.1. \(\text{left}_m \text{left}_n p = \text{left}_{J(p, m, n)} p\) if \(\text{left}_n p\) is defined.

Proof. If \(n = 0\) then \(J(p, m, n) = m\) and \(\text{left}_n p = p\); thus \(\text{left}_{J(p, m, n)} p = \text{left}_m \text{left}_n p\).

Now induct on \(n\). Assume \(\text{left}_{n+1} p\) is defined. Note that \(n + 1 \leq \text{len} p\) by Theorem 6.1, so \(p \neq ()\). I will show that \(\text{left}_m \text{left}_{n+1} p = \text{left}_{J(p, m, n+1)} p\).

1. Say \(p = qC\). Write \(j = J(q, m, n)\). Then

\[
\text{left}_m \text{left}_{n+1} p = \text{left}_m \text{left}_n q = \text{left}_j q = \text{left}_{j+1} p.
\]

2. Say \(p = q\bar{r}\) and \(n < \text{len} r\). Write \(j = J(q\bar{r}r, m, n)\). Observe that

\[
\text{left}_m \text{left}_{n+1} p = \text{left}_m \text{left}_n q \bar{r}r = \text{left}_j q \bar{r}r.
\]

If \(j < \text{len} r\) then \(\text{left}_j q \bar{r}r = \text{left}_{j+1} p\); else \(\text{left}_j q \bar{r}r = \text{left}_{j-\text{len} r} q \bar{r}r = \text{left}_{j-\text{len} r} p\).

3. Say \(p = q\bar{r}\) and \(n \geq \text{len} r\). Write \(j = J(q, m, n - \text{len} r)\). Then

\[
\text{left}_m \text{left}_{n+1} p = \text{left}_m \text{left}_{n-r} q = \text{left}_j q = \text{left}_{j+\text{len} r+1} p.
\]

4. Say \(p = q(r' \cup r)\) and \(n < \text{len} r\). Write \(j = J(qr, m, n)\). Observe that

\[
\text{left}_m \text{left}_{n+1} p = \text{left}_m \text{left}_n q r = \text{left}_j q r.
\]

If \(j < \text{len} r\) then \(\text{left}_j q r = \text{left}_{j+1} p\); otherwise

\[
\text{left}_j q r = \text{left}_{j-\text{len} r} q = \text{left}_{j-\text{len} r+\text{len} r'} q r' = \text{left}_{j+\text{len} r'+1} p.
\]
5. Say $p = q(r' \cup r)$ and $n \geq \text{len} r$. Write $j = J(qr', m, n - \text{len} r)$. Then
\[
\text{left}_m \text{left}_{n+1} p = \text{left}_m \text{left}_{n-\text{len} r} qr' = \text{left}_j qr' = \text{left}_{j+\text{len} r+1} p.
\]

\[\square\]

**Theorem 7.2.** If $p'$ is a left subpattern of $p$ then $p' = \text{left}_n p$ for some $n$.

**Proof.** If $p = qC$ then $q = \text{left}_1 p$.

If $p = q\overline{r}$ then $q = \text{left}_{\text{len} r+1} p$. Furthermore $q\overline{r}r = \text{left}_1 p$ if $r \neq ()$; $q\overline{r}r = \text{left}_0 p$ if $r = ()$.

If $p = q(r' \cup r)$ then $qr' = \text{left}_{\text{len} r+1} p$. Furthermore $qr = \text{left}_1 p$ if $r \neq ()$. If $r = ()$ then $qr = q = \text{left}_{\text{len} r} qr' = \text{left}_{\text{len} r' + 1} p$ by Theorem 6.2.

Finally, by Theorem 7.1, the relation “$p'$ is $\text{left}_n p$ for some $n$” is transitive. \[\square\]

**References**


Department of Mathematics, Statistics, and Computer Science, The University of Illinois at Chicago, Chicago, IL 60607–7045

E-mail address: djb@pobox.com